EE 660 Project: PUBG Finish Placement Prediction

Abstract: *In this project, I participated in the Kaggle competition PUBG Finish Placement Prediction. The objective is to try to get the best prediction on how a player will place given some features, using regression. I explored the data, performed preprocessing on the inputs, performed baseline regression using linear regression. Then, I performed cross-validation to get the best parameters for the XGBoost model. Finally, I tested the model and interpreted the results.*

Introduction

PlayerUnknown's BattleGrounds (PUBG) is a popular Battle Royale-style video game where the objective is to eliminate other players on a map until one is left standing. The team at PUBG has made official game data available, split into training and testing sets, and in this competition, we are asked to predict final placement from final in-game stats and initial player ratings [1]. We are initially given 28 features, and our task is: given those features, predict a player's placement as a percentage from 0 (last place) to 1 (first place). The criterion for evaluation is Mean Absolute Error (mae) between the predicted value and the observed value.

This is a regression problem, where we want to predict a continuous value given input variables. This is a good machine learning problem, because we are given 28 features, and looking at the data intially, unless we were an expert at this game, we may not know how to predict how a player will perform. My primary objective throughout this whole project is to create a model that gives the best predictions on the data. First, I will describe the PUBG player dataset. Next, I will apply a preprocessing technique called standard scaling. I will discuss the dataset methodology I will use for training, validation, and testing. Next, for training, I will I will apply linear regression as a baseline model, and then I will apply XGBoost, which has a track record of high performance and high flexibility, on the data. Then, I will use cross-validation to select the best parameters for the XGBoost model. I will interpret describe the final model, and then I will test the model on test data.

Implementation

Data Set: I will now explain the data set used in this competition. There are 4446966 data points in the test set, 1934174 data points in the test set, with 24 numerical features, and 4 categorical features. The table on the next page describes the features.

feature	type	range / # of categories	description
Id	categorical	unique to each player	Player's Id
groupId	categorical	2 million different groups	ID to identify a group within a match
matchId	categorical	48000 different matches	ID to identify a group within a match
assists	numeric	0 - 22	Number of enemy players this player damaged that were killed by teammates
boosts	numeric	0 - 33	Number of boost items used
damageDealt	numeric	0 - 6616	Total damage dealt
DBNOs	numeric	0 - 53	Number of enemy players knocked
headshotKills	numeric	0 - 64	Number of enemy players killed with headshots
heals	numeric	0 - 80	Number of healing items used
killPlace	numeric	1 - 101	Ranking in match of number of enemy players killed
killPoints	numeric	0 - 2170	Kills-based external ranking of player
kills	numeric	0 - 72	Number of enemy players killed
killStreaks	numeric	0 -20	Max number of enemy players killed in a short amount of time
longestKill	numeric	0 - 1094	Longest distance between player and player killed at time of death
matchDuration	numeric	9 - 2237	Duration of match in seconds
matchType	categorical	16 different match types	String identifying the game mode that the data comes from
maxPlace	numeric	1 - 100	Worst placement we have data for in the match
numGroups	numeric	1 - 100	Number of groups we have data for in the match
rankPoints	numeric	-1 - 5910	Elo-like ranking of player
revives	numeric	0 - 39	Number of times this player revived teammates
rideDistance	numeric	0 - 40710	Total distance traveled in vehicles measured in meters
roadKills	numeric	0 - 18	Number of kills while in a vehicle
swimDistance	numeric	0 - 3823	Total distance traveled by swimming measured in meters
teamKills	numeric	0 - 12	Number of times this player killed a teammate
vehicleDestroys	numeric	0 - 5	Number of vehicles destroyed
walkDistance	numeric	0 - 25780	Total distance traveled on foot measured in meters
weaponsAcquired	numeric	0 - 236	Number of weapons picked up
winPoints	numeric	0 - 2013	Win-based external ranking of player

Table 1: 28 features are outlined in detail

Preprocessing: Looking at the data, the numeric features have different ranges, so it is best to scale each feature to have 0 mean and 1 variance. First, I scaled the features of the training set, and then, I applied the same scaling parameters to the test set. Next, I converted matchType to numerical data using one-hot encoding, giving 16 different columns. Then, I removed Id, which is not useful data. Then, I converted groupId and matchId into category codes, where each unique value gets a different category code. I used this for groupId and matchId because it is much more efficient than one-hot encoding for large numbers of categories. I performed the same categorical conversion on the test data. Now, we have all numerica data that is properly scaled, so we can start training.

Dataset Methodology: I used two different dataset methodologies for the two different regression models. For linear regression, I made a 70/30 split into train and validation sets on the original train set. This is because I wanted to test the baseline model and quickly get a validation score that we can compare with later. Next, for the XGBoost model, I used 3-fold cross-validation so that I could perform model selection and also use all of the training data. The cross-validation loop was performed for each parameter at a time, unlike grid search, which is much more computationally efficient, but may have less performance.

For the test set, I performed the same preprocessing as the training data, without using any information from the test set. I saved the test set until the very end, so there was no data snooping. The test set was used to create submissions for the Kaggle competition after creating the model, at the very end of the process.

Training Process: I first used linear regression as a baseline model. Linear regression is a model where the input and output are assumed to have a linear relationship [2]:

$$y = wx$$

Where the weights, w, are to be learned. I trained the model using the training data, and then I used the validation set to test the model. The mean absolute error for the validation data was 0.0906.

Next, I used XGBoost, which stands for eXtreme Gradient Boosting. Boosting is a greedy algorithm for fitting adaptive basis-function models. An adaptive basis-function model is of the form [3]:

$$f(x) = w_0 + \sum_{m=1}^{M} w_m \phi_m(x)$$

where the $\phi_m(x)$ are "weak learners" are very simple linear regression trees. In order to fit this adaptive basis function, we minimize the loss:

$$\min_f \sum_{i=1}^{N} L(y_i, f(\mathbf{x}_i))$$

where L is some loss function. The XGBoost model uses squared error. Now, in Gradient Boosting, to find the optimal function to minimize the loss, we will use stagewise gradient descent, where at each step m [4],

$$g_{im} = \left[\frac{\partial L(y_i, f(\mathbf{x}_i))}{\partial f(\mathbf{x}_i)} \right]_{f=f_{m-1}}$$

And we then make the update

$$\mathbf{f}_m = \mathbf{f}_{m-1} - \rho_m \mathbf{g}_m$$

where ρ_m is the learning rate.

XGBoost has a track record of high performance on Kaggle competitions. It is also very flexible. XGBoost allows for regularization, which can help with overfitting. It is highly flexible. It allows for cross-validation at each iteration of the boosting process and makes it easy to get the exact optimum number of boosting iterations in a single run [5].

In order to get the optimal parameters of the XGBoost model, I will do 3-fold cross validation. The parameters I will optimize are eta (learning rate), max_depth (maximum depth of decision tree), min_child_weight (minimum sum of instance weight (hessian) needed in a child), subsample (subsample ratio of the training instances), lambda(L2 regularization term on weights), and alpha (L1 regularization term on weights). Eta is the learning rate of each boosting step. Smaller eta will make make new weights smaller. Min_child_weight helps prevent overfitting by preventing small children nodes. It is pruning those children nodes that might "memorize" the data. Subsample randomly samples a ratio of the training data prior to growing trees, and this will prevent overfitting. Lambda and alpha correspond to L2 and L1 regularization, respectively, which help to prevent overfiting.

After doing 3-fold cross-validation on the train data using 10 boosting rounds, the optimal parameters are:

eta = 0.28, max_depth = 5, min_child_weight = None, subsample = 1, lambda = 0.01, and alpha = None.

It looks like the parameters min_child_weight and alpha are not used, so they don't help with performance Lambda does not help either, since it is only a small value and gives tiny L2 regularization. Subsample is 1, so it is not used. So, all of the parameters that are supposed to help with overfitting are pretty much not used in this model. ***We can reason that we are underfitting the data,*** and that we have room to increase the complexity of our model, due to the fact that we have an enormous number of training data points and a relatively small number of features. We can increase the number of boosting rounds and the max_depth of our decision trees to increase cross-validation performance, but due to the computational limitations of the Kaggle kernel and due to limited time, I will not proceed. After cross-validation, the optimal parameters gave the best mae = 0.079.

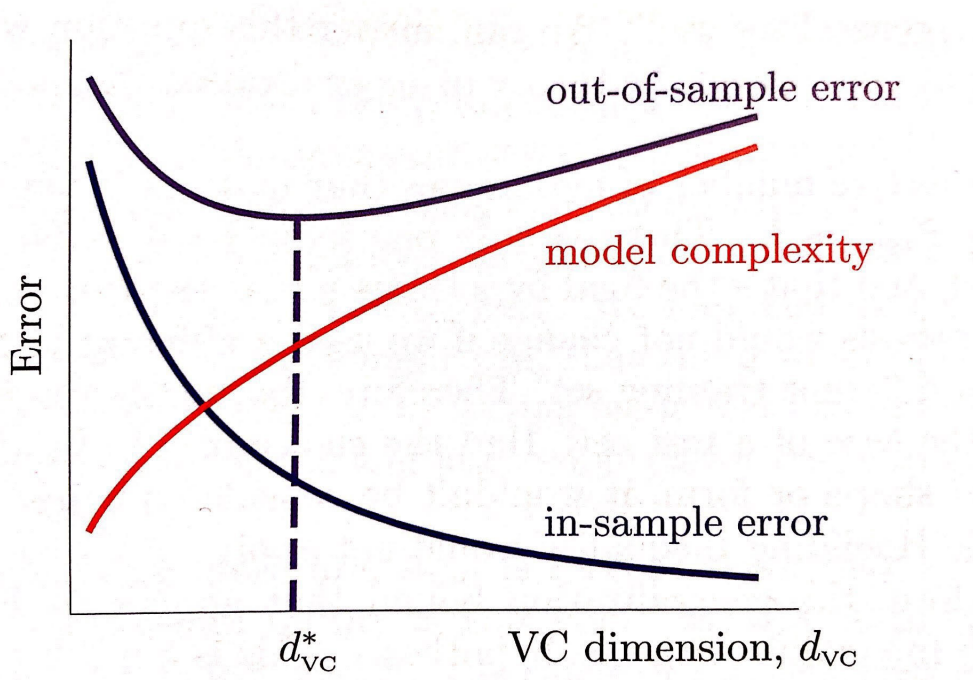

Figure 1: showing the relationship between the model complexity and the in-sample and out-of-sample error [6]

In the graph above, we see how model complexity relates to our error. For our model, we are likely to the left of the optimal d_{VC}. We can increase the model complexity and further reduce both the in-sample and out-of-sample error. After we reach the optimal VC dimension or model complexity, we have the lowest out-of-sample error. But again, due to limitations of space and time, I cannot proceed.

Results: The final training models are shown:

model	validation mae
linear regression	0.0906
XGBoost	0.079

Table 2: final validation mae for our models

XGBoost performs better than linear regression, so we will use XGBoost for testing. Note that linear regression had pretty good performance. We can say that our data has a lot of linear relationships that the linear regression was able to capture. But the objective is to get higher performance, and XGBoost will capture non-linear relationships. Then, I trained a final XGBoost model using the optimal params and all of the training data, with 10 boosting rounds, the same number of rounds used in cross-validation. Increasing the number of boosting rounds in our final model is likely to increase the test accuracy, but we can't data snoop and must base our model based on what we know.

Final Results and Interpretation

Now, I will use the trained XGBoost model with the optimal parameters on the test data. The mae of the test data is 0.061. This is surprising that the testing mae is *lower* than the validation mae of the optimal XGBoost model. ***What this is telling us is that during cross-validation, we were underfitting on our validation set, but in our final trained model, we learned enough to generalize well on the test set.*** This is due to having an enormous training set that we used during cross-validation. As a quick test, I sampled the training set to a smaller value, and after doing cross-validation again, the validation mae went down. What we could do is do CV on a smaller sampled training set to select our parameters, but that may give less optimal parameters compared to doing CV on the entire training set, so I did not do that. The computational limit when doing cross-validation on the training data proved to be a challenge for this project, and we had to limit the number of boosting rounds because of time limits, causing underfitting. The final model also used the same number of boosting rounds, so it also likely underfits. Future work would explore this further and attempt to create a more complex model by increasing the number of boosting rounds, or other methods, in order to increase the final model's complexity and get even higher test performance.

Using the test data, we can bound the out-of-sample error using the generalization bound discussed in Abu-Mostafa et al [7]:

$$E_{out}(h_g) \leq E_{in}(h_g) + \sqrt{\frac{1}{2N} ln \frac{2M}{\delta}}$$

where M = 1, δ = 0.1, N = 1934174, and in-sample error $E_{in}(h_g) = 0.610$. I calculated that $E_{out}(h_g) \leq 0.6109$. That is a very tight bound on our in-sample error! This is because we used only 1 hypothesis, and we have an enormous amount of test data.

The score of 0.0610 would place me at 535 on the public leaderboard of the Kaggle PUBG competition. That means that I have a lot of room of improve to increase my model's performance. Looking at other Kaggle kernels, great improvements came from very selective feature engineering, which I did not due, since I decided to focus on the machine learning aspects of this project. Future work would include feature engineering and data exploration in order to improve our model.

Conclusion

In this project, I competed in the PUBG Finish Placement Prediction competition. I selected the XGBoost model initially as the model I will tune and train on because of its track record of performance on other Kaggle competitions. After tuning the parameters of the model using cross-validation, I trained a final model on all of the training data, and this model got a mae score of 0.0610.
Due to the computational limitations on the Kaggle kernel, I had to limit the complexity of the final trained model, causing underfitting and likely limiting the performance on the test data. Future work would include feature engineering and increasing the complexity of our model to improve final performance.

References:

[1]PUBG Finish Placement Prediction, Kaggle https://www.kaggle.com/c/pubg-finish-placement-prediction Accessed: 12/2/2018.
[2] Murphy, Kevin P. Machine Learning: a Probabilistic Perspective. MIT Press, 2013. P. 19
[3] Murphy, Kevin P. Machine Learning: a Probabilistic Perspective. MIT Press, 2013. P. 543
[4] Murphy, Kevin P. Machine Learning: a Probabilistic Perspective. MIT Press, 2013. P. 560
[5] Complete Guide to Parameter Tuning in XGBoost (with codes in Python), Analytics Vidhya. https://www.analyticsvidhya.com/blog/2016/03/complete-guide-parameter-tuning-xgboost-with-codes-python/ Accessed: 12/2/2018.
[6] Abu-Mostafa, Yaser S., et al. Learning from Data: a Short Course. AMLbook, 2012. P. 59

Thank you for an excellent semester!